Technical Report: TR-96-2, March 07, 1996

Latency and Bandwidth Requirements of Massively Parallel Programs: FFT as a Case Study

Petrini,Fabrizio Vanneschi,Marco

Dipartimento di Informatica, Università di Pisa
corso Italia 40, 56125 Pisa, Italy

Abstract

Many theoretical models of parallel computation are based on overly simplistic assumptions on the performance of the interconnection network. For example they assume constant latency for any communication pattern or infinite bandwidth. This paper presents a case study based on the FFT transpose algorithm, which is mapped on two families of scalable interconnection networks, the k-ary n-trees and the k-ary n-cubes. We analyze in depth the network behavior of a minimal adaptive algorithm for the k-ary n-trees and three algorithms for the k-ary n-cubes, each offering an increasing degree of adaptivity: the deterministic routing, a minimal adaptive routing based on Duato's methodology and the Chaos routing, a non-minimal adaptive cut-through version of the hot potato routing. The simulation results collected on topologies with up to 256 processing nodes show that the k-ary n-trees can efficiently support the basic FFT algorithm by factoring the personalized broadcast in a sequence of congestion-free steps. Though k-ary n-cubes are less favored in terms of bisection bandwidth, we can narrow the performance gap between the two interconnection networks by properly interleaving communication and computation. While in the presence of bandwidth-bound patterns the communication latency becomes difficult to predict, the global accepted network bandwidth converges to a fixed value after a stabilization period, though both adaptive algorithms on the cubes suffer from post-saturation problems.

ACM–CR Subject Classification:

Keyword and phrases: Fast Fourier Transform, Interconnection Networks, Parallel Algorithms, Parallel Architectures, Routing, Wormhole

1 Introduction

The computation of the Fast Fourier Transform (FFT) of a vector of complex numbers plays an important role in several scientific and technical fields. Some of the applications of the FFT algorithm include signal processing, solving linear partial differential equations, convolution, integer division and multiplication, polynomial interpolation, etc. [Lei92]. Also, the computational dependence graph of the FFT, called *butterfly* graph, has been used in the interconnection network of tightly coupled parallel machines, due to its symmetries and scalability properties [Hwa93] and is a basic block of the counting and bitonic sorting networks [AHS93]. For these reasons the implementation of FFT on parallel computers has raised a great interest in the scientific community over the last decade [GK93, Chu87, Bai90, Bai87, Bai88, Swa87, Swa84, Get92, RWW91].

In this paper we analyze through simulation the characteristics of the communication patterns generated by some parallel FFT algorithms on two popular families of interconnection networks, the k-ary n-cubes [Dal90] and the k-ary n-trees [PV95a], using wormhole and cut-through routing algorithms. The goal is to verify the gap between theoretical models of the interconnection networks and actual performance results.

Wormhole is a flow control protocol that divides each packet in elementary units called *flits* and advances each flit as soon as it arrives at a node in a pipeline fashion. Wormhole is attractive because it reduces the latency of message delivery compared to store and forward and requires only a few flit buffers per node. Deadlock can be avoided and throughput of wormhole networks can be increased by organizing the flit buffers associated with each physical channel into several virtual channels [Dal92]. These virtual channels are allocated independently to different packets and compete with each other for the physical bandwidth, allowing active messages to pass blocked messages using network bandwidth that would otherwise be wasted. Cut through uses the same pipeline strategy of wormhole at the flit level, but in this case each intermediate node between source and destination provides enough buffer space to store the whole packet in case of blocking, once the trailing flit has been accepted [PV95b].

The experimental results collected on topologies with up to 256 nodes compare three routing algorithms on the k-ary n-cubes (deterministic, minimal adaptive and fully adaptive) and a minimal adaptive algorithm on the k-ary n-trees.

The deterministic algorithm on the k-ary n-cubes forces the communication to take place on a unique path, even when multiple paths are available and is based on the dimension order routing. Deadlock is avoided on the wrap around connections associating two virtual channels to each physical link, that are allocated according to an acyclic buffer dependency graph [DS87]. The minimal adaptive routing limits the path selection to the shortest paths between source and destination and is based on Duato's methodology, that requires the absence of cyclic dependencies on a connected channel subset [Dua94]. In our version there are three virtual channels for each link. The non minimal algorithm is the Chaos routing, a cut through version of the hot potato routing. The Chaos router behaves as a normal minimal router under normal traffic conditions and stores messages in an internal buffer, the multiqueue, in the presence of high load, resorting to a randomized algorithm to deroute the packets and potentially choosing a non minimal intermediate node when the multiqueue is full [Bol93].

Minimal adaptive routing between a pair nodes on a k-ary n-tree can be accomplished sending the packet to one of the nearest common ancestors (NCA) and from there to the destination. That is, each packet experiences two phases, an adaptive ascending phase to get to an NCA, followed by a deterministic descending phase. This algorithm uses a single virtual channel [PV95a].

Understanding the real performance of an interconnection network and its routing algorithm is a key factor to design scalable and portable parallel algorithms. Many algorithms are designed having in mind factors not present in any reasonable machine, such as zero communication delay or infinite bandwidth. The frequently used PRAM model [FW78] is simple and useful for a gross classification of algorithms, but it does not reveal important performance bottlenecks in the interconnection network. As a consequence, a variety of alternative models have been developed, each capturing different aspects of real parallel machines. The Delay Model [PY88] considers the communication latency, realizing that there is a delay between the time some information is produced at a processor and the time it can by used by another processor. Another model that addresses communication latency is the BPRAM [ACS89], in which block transfers are allowed. A model that deals with limited communication bandwidth is the LPRAM [ACS90].

Some models address several of the above aspects together. For example, Valiant's BSP [Val90], the Postal Model [BNK92] and LogP [CKP+93a] include both latency and communication bandwidth through parameters. The BSP describes a parallel machine in terms of three elements: processor-memory modules, an interconnection network and a synchronizer, which performs barrier synchronization. A computation consists of a sequence of *supersteps*. During a superstep each processor performs local computation and receives and sends messages subject to the following constraints: the local computation may depend only on data present in the local memory of the processor at the beginning of the superstep. A processor may send at most h messages and receive at most h messages in a superstep. Valiant provides a theoretical analysis showing that the simulation of algorithms designed for the PRAM model on the BSP is optimal, up to constant factors, provided that the *parallel slackness*, i. e. the number of PRAM processors per BSP processor, is sufficiently large. However the constant factors in the simulation may be large and would be even larger if the cost of context switching and cache misses were fully counted. Both the Postal Model and LogP try to ignore the exact structure of the network and assume that it creates a complete communication graph between the processors. Such systems consider sending a message as a send-and-forget event and assume that passing a message between any pair of processors takes roughly the same time. LogP characterizes a parallel machine through the use of four parameters: the communication latency (L), overhead (o), bandwidth (g) and the number of processors (P). Furthermore, it is assumed that the network has finite capacity, such that at most $\lceil L/g \rceil$ messages can be in transit from any processor or to any processor at any given moment during program execution. Some interesting applications of LogP are shown in [KSS92, Sah92, AISS95, CLMY96].

The remainder of this paper is organized as follows. Section 2 presents three sequential FFT algorithms and some properties of the butterfly graph that are used in section 3 to implement two variants of the transpose parallel FFT algorithm. While the first one strictly separates the computation of the butterfly with a global personalized broadcast, the second one interleaves computation and communication and exploits the internal parallelism of the processing nodes. Section 4 describes the relevant details of the simulation model that we used in Section 5 to evaluate the performance of the FFT algorithms on two classes of interconnection networks, the k-ary n-trees and the k-ary n-cubes. The experiments analyze in depth the impact of several routing algorithms on the network performance. An overview of the experimental results is provided in section 6 and some concluding remarks are given in section 7.

2 Sequential FFT algorithms

The FFT developed by Cooley and Tukey in the mid-60s is a method of computing the discrete Fourier transform which reduces the number of operations for an n-point complex vector from $O(n^2)$ to $O(n \log n)$. The data-flow graph induced by an n-point FFT computation is called butterfly graph and is a directed graph with $n(\log n + 1)$ nodes arranged in n rows and $\log n + 1$ columns. Nodes are labeled by ordered pairs (r, c), with $0 \leq r < n$ and $0 \leq c \leq \log n$. For $0 \leq r < n$ and $0 \leq c < \log n$, node (r, c) has a direct connection to nodes $(r, c+1)$ and $(\bar{r}_c, c+1)$, where r and $\bar{r}_c$ differ only in the $(c+1)$-th most significant bit of their binary representation. The nodes in column 0 are the problem inputs and those in column $\log n$ are the outputs. Each non-input node represents an atomic computation.

The butterfly graph can be partitioned into two groups of sub-butterflies, each having the same dimension. This property can be used to implement several variants of sequential and parallel algorithms [Sah92, Chu87]. The following corollary describes how to partition a given butterfly.

Corollary 2.1 *Given an n-input*[1] *butterfly graph*

- *for each* $0 < j \leq \log n$*, the nodes in columns* 0 *through* j *comprise* $\frac{n}{2^j}$ *disjoint* 2^j*-input butterflies with the i-th butterfly being*

$$A_i = \{(r,c) : r \equiv i \ (mod\ \frac{n}{2^j}),\ 0 \leq c \leq j\},\ 0 \leq i < \frac{n}{2^j}; \tag{1}$$

- *for each* $0 \leq k < \log n$ *the nodes in columns* k *through* $\log n$ *comprise* 2^k *disjoint* $\frac{n}{2^k}$*-input butterflies, with the i-th butterfly being*

$$B_i = \{(r,c) : i\frac{n}{2^k} \leq r < (i+1)\frac{n}{2^k},\ k \leq c \leq \log n\},\ 0 \leq i < 2^k. \tag{2}$$

[1] We will assume throughout the paper that n is a power of two.

The computation of an n-input butterfly can be organized in three distinct phases. Given $0 < j < \log n$, in the first phase we compute the non-input nodes belonging to the initial j columns. Data are properly rearranged in the second phase and the third phase computes the nodes in the remaining $\log n + 1 - j$ columns.

As shown in Corollary 2.1, the first j columns include $\frac{n}{2^j}$ distinct 2^j-input butterflies. If we consider the nodes in column j as the input nodes of the remaining $\log n + 1 - j$ columns, we can easily recognize 2^j independent $\frac{n}{2^j}$-input butterflies. Defining A_i, $0 \leq i < \frac{n}{2^j}$, the butterflies included in the first j columns and B_k, $0 \leq k < 2^j$ the remaining butterflies, there is a direct connection in the resulting data flow graph between the k-th output node of A_i and the i-th input node of B_k. That is, there is one (and only one) edge connecting any two butterflies on opposite sides of the partition.

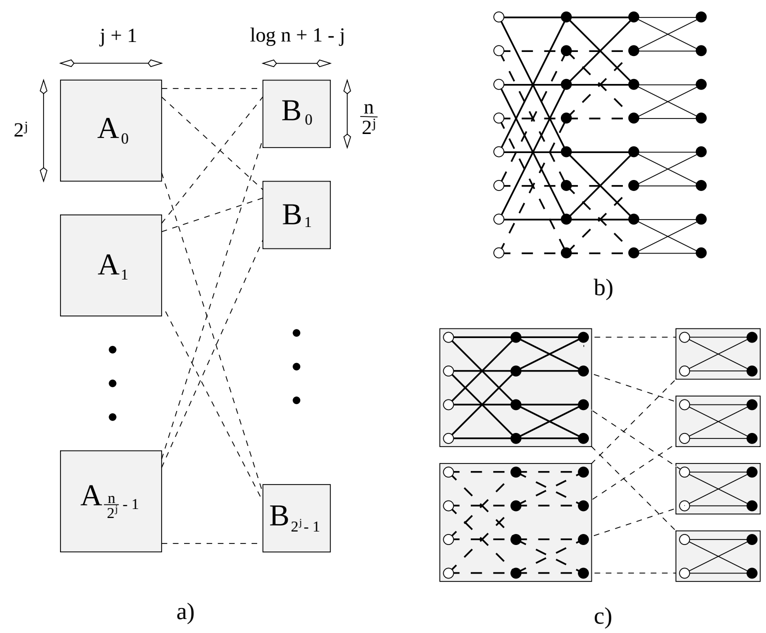

Figure 1: Butterfly partition. a) Overview. b) A 8-input butterfly is partitioned choosing $j = 2$. c) The partition cuts the original butterfly in $2 + 4$ butterflies.

The general structure of the partition is shown in Figure 1 a); as an example, the computation of a 8-input butterfly can be scheduled by 2 distinct 4-input butterflies followed by 4 distinct 2-input butterflies as reported in Figure 1 b) and 1 c).

In our analysis we will consider three sequential algorithms, based on different scheduling of the butterfly nodes, defined

- *naive*,
- *hierarchical*,
- and *cache oriented*.

In the naive algorithm the butterfly nodes are computed column-wise, starting from column 1 till column $\log n$ and row-wise inside each column. This computation can be described with two nested loops, the outer one computing $\log n$ columns and the inner one n rows.

An alternative scheduling, defined hierarchical, exploits the recursive structure of the butterfly. It can be shown that a n-input butterfly can be partitioned into two $\frac{n}{2}$-input sub-butterflies, removing the level 0 nodes [Lei92]. This algorithm visits in a recursive way the graph, as shown in Figure 2

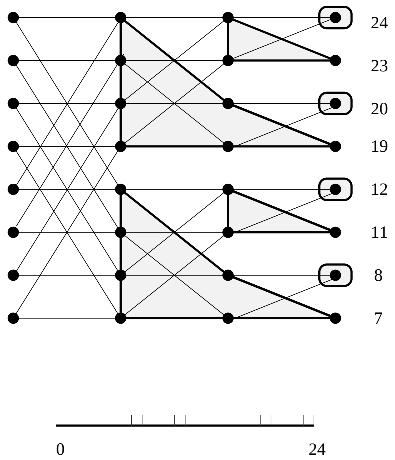

Figure 2: An example of hierarchical scheduling on a 8-input butterfly. Assuming that the computation of each internal node requires a single unit time, the bottom right output node $(0, 3)$ is computed at time 7, node $(1, 3)$ at time 8 and so on. The output times are outlined in the time scale under the butterfly.

While the computation of the output nodes in the naive algorithm is left to the final part, in the hierarchical algorithm the values of the output nodes are available in a more interleaved way. This will be useful to overlap communication and computation in a parallel algorithm shown in the following sections. For example, the value of node $(0, 3)$ is available at time 7 in the example shown in Figure 2 and at time 17 using a naive scheduling.

The third algorithm, called cache-oriented, uses the partition property shown in Corollary 2.1. This algorithm tries to optimize the memory hierarchy utilization, partitioning the butterfly in sub-butterflies whose working set can be fully contained in the cache memory. Basically, we find a suitable value of j and we recursively apply the partition until each butterfly has a number of inputs equal or less than 2^j. The resulting graph has $\lfloor \frac{n}{j} \rfloor$ stages of 2^j-input butterflies and adjacent stages are connected according to an omega topology. On their turn, the basic butterflies can be scheduled using the naive or the hierarchical algorithm.

3 Parallel FFT algorithms

The most commonly used parallel algorithm maps each row r on node $\lfloor \frac{rP}{n} \rfloor$, where P is the number of nodes[2]. With this mapping, nodes need to communicate with each other during the computation of the first $\log P$ columns. The remaining $\log \frac{n}{P}$ columns are available on the same node. While this mapping is effective on the hypercubes because generates a *normal* algorithm [Lei92], low-dimensional cubes cannot make efficient use of a large number of processing nodes [GK93].

An alternative formulation, often named *transpose* algorithm, uses Corollary 2.1 to divide the computation into two computational phases separated by a global communication phase. Assuming that the data set is big enough, that is $n \geq P^2$, we choose $j = \log \frac{n}{P}$. Each node computes in the first phase a $\frac{n}{P}$-input butterfly available locally, allocated according to a cyclic mapping. In the second computational phase each node has assigned $\frac{n}{P^2}$ butterflies[3], whose size is P. As already noted, there is a direct connection between any two butterflies belonging to different computational phases. This implies that the communication phase is an *all-to-all personalized broadcast* (AAPB) where each processing node at the end or during the first computational phase has to send $P-1$ messages containing the values of $\frac{n}{P^2}$ output nodes. In our experiments we will consider two variants of the transpose algorithm, called

- *basic*
- and *overlapped.*

While in the basic variant the first phase and the global communication are strictly separated, in the overlapped algorithm we try to interleave communication and computation in order to exploit the internal parallelism available in many existing and foreseeable parallel machines. The first phase uses a cache oriented sequential algorithm whose basic butterflies are computed with the hierarchical version. The goal is to make available as soon as possible the values of the output nodes, that can be sent to the other processing nodes before the end of the local computation.

4 Relevant details of the simulated model

This section presents the simulation model of parallel architectures that is used to analyze the behavior of the FFT algorithms. We focus our attention on two main classes of scalable interconnection networks, the k-ary n-cubes and k-ary n-trees, described in more detail below. In order to capture the real behavior of the interconnection network, the model describes several routing algorithms at the flit level. Each processing node is equipped with a communication processor that off-loads the main processor from remote message handling. Also, the model provides a detailed description of the memory hierarchy inside the processing nodes.

[2] As with the butterfly dimension, we will assume that P is a power of two.

[3] If both n and P are powers of two, the same holds for $\frac{n}{P^2}$.

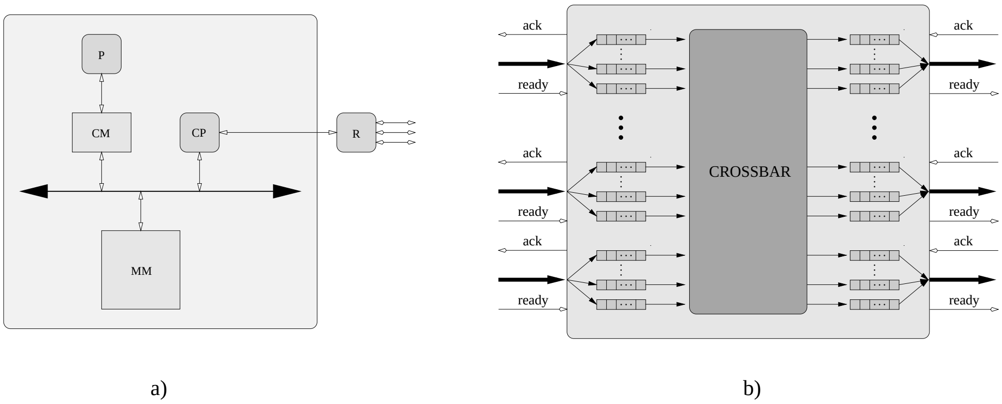

Figure 3: a) The internal structure of the processing node. b) The internal structure of the router. P processor, CM cache memory, MM main memory, CP communication processor, R router.

4.1 Processing node

The internal structure of the processing node is shown in Figure 3 a) and is centered around the main bus. We distinguish the following units:

- the *main processor*,
- its *cache memory*,
- the *main memory*,
- the *communication processor*
- and the *router*.

The computational part of the algorithms is mapped on the main processor. We assume that the computation of the non-input nodes of the butterfly, about 10 floating point operations, requires $T_{compute}$ clock cycles when the operands are on chip.

While the processing power of general-purpose commodity processors is racing to high levels, the building of memory systems that can keep pace becomes increasingly problematic. The presence of one or more levels of cache memories is a common solution to smooth the performance gap between aggressive processors and main memory. For this reason the main processor has associated a cache memory with the following parameters: N_{sets} the number of sets, N_{set_size} the number of cache lines in each set, N_{line_size} the dimension, in bytes, of the cache line. The overall cache capacity in bytes is

$$N_{cache\_size} = N_{sets} * N_{set\_size} * N_{line\_size}. \tag{3}$$

When the main processor reads from the cache, it gets a hit in T_{read} cycles and can write in T_{write} cycles. Writes are non blocking, but there is a buffer between processor and cache that limits the number of outstanding writes. The cache loads the corresponding cache line in the presence of both read or write misses. Read misses are served in T_{read_miss} cycles, if there is no contention on the main bus.

The computational processor is dedicated to message handling and acts as a Message Driven Processor [D^+92]. It fragments outgoing messages in packets that are enqueued in a FIFO directed to the router and receives incoming packets from another FIFO. The time spent in packet handling is T_{packet} cycles, both in sending and receiving a packet. The communication processor can perform DMA-like activities and store the content of incoming messages in proper memory locations. There is a simple coherency mechanism between the cache memory and the communication processor. The cache controller snoops on the main bus, and every time a memory location is overwritten by the communication processor, it invalidates the corresponding cache line, if present in the cache memory. Symmetrically, when the communication processor tries to read a memory location replicated on the cache memory, it gets the updated copy. Both processor can send interrupts that are served in $T_{interrupt}$ cycles.

4.2 Router

Figure 3 b) outlines the internal structure of a router. We can distinguish the external channels or links, the input and the output buffers or lanes that implement the buffer space of the virtual channels and an internal crossbar. Each channel on the single direction is logically composed of three interfaces: a *data path* that transmits messages on a flit level, a *ready* line that flags the presence of a flit on the data path and specifies the virtual channel where the flit is to be stored and an *ack* line in the reverse direction that sends an acknowledgment every time buffer space is released in the input lanes.

A flit is moved from an output lane to the corresponding input lane in a neighboring node in T_{link} cycles, when there is at least a free buffer position. Each output lane has associated a counter that is initialized with the total number of buffers in the input lane, it is decremented after sending a flit and it is incremented upon receiving and acknowledgment. When multiple lanes are enabled, an arbiter picks one of them according to a fair policy. When the flit is put on the data path, the ready line signals the virtual channel where the flit must be stored. Assuming v virtual channels for each link, the ready line can be coded with $\lceil \log_2 v \rceil$ bits.

When a header flit reaches the top of an input lane, the routing algorithm tries to establish a path in the crossbar with a suitable output lane, that is neither full nor bound to another input lane. This path will remain in action till the transmission of the tail flit of the packet. Our model allows the routing of a single header at a time every $T_{routing}$ cycles. The extra complexity of a parallel router has been shown to give little or no advantage in terms of performance in the presence of wormhole or cut-through flow control [Bol93].

Although each physical link services at most one virtual channel in each flit cycle, multiple virtual channels can be active at the input and output ports of the crossbar. Every time a flit is moved from an input lane to the corresponding output lane, a feedback is sent back to the neighboring switch or node to update the counter of free positions. According to this model, given two nodes p and q with $p \neq q$, the *base network latency* $T_{lat}(p,q)$ to send a packet in the absence of contention can be defined as

$$T_{lat}(p,q) = (d(p,q)-1)T_{routing} + (d(p,q)+P_{length}-1)T_{link}, \qquad (4)$$

where P_{length} is the packet length in flits and $d(p,q)$ the topological distance between nodes p and q.

4.3 Network topology and routing algorithms

Quaternary fat-trees and bi-dimensional tori are emerging standards in the design of interconnection networks for parallel machines and are the representatives of the classes of k-ary n-trees and k-ary n-cubes that we have chosen in our experiments.

Minimal adaptive routing between a pair nodes on a k-ary n-tree can be easily accomplished sending the packet to one of the common roots or *nearest common ancestors* (NCA) of sender and destination and from there to the destination. That is, each message experiences two phases, an *ascending* phase to get to one of the NCA, followed by a *descending* phase [PV95a]. Our version of this algorithm uses a single virtual channel for each physical link.

Routing algorithms on the k-ary n-cubes are deadlock-prone and require more sophisticated strategies. We compare three algorithms, each offering a different degree of adaptivity:

- *deterministic*,
- minimal adaptive based on *Duato*'s methodology,
- non minimal adaptive, the *Chaos* router.

The deterministic algorithm is a dimension order routing based on a static channel dependency graph [DS87]. Packets are sent to their destination along a unique, predetermined and minimal path. The potential deadlocks caused by the wrap-around connections are avoided doubling the number of virtual channels and creating two distinct virtual networks. Packets enter the first virtual network at the beginning and switch to the second network upon crossing a toroidal connection. Our version uses two virtual channels for each physical link.

Rather than using a static channel dependency graph, Duato's methodology only requires the absence of cyclic dependencies on a connected channel subset. The remaining channels can be used in almost any way. We associate three virtual channels to each link: on one of these channels packets are routed along any minimal path between source and destination. The remaining two channels are "escape" channels where packets are routed deterministically when the adaptive choice is limited by network contention. A static channel dependency graph is used in the channel subset [Dua94].

The Chaos router is a non minimal cut-through routing algorithm. Under normal conditions, it routes packets from its input buffers to the output buffers as in minimal adaptive routing. The Chaos router has a central multiqueue to avoid blocking and deadlocks. A packet is moved into the multiqueue if it is potentially in a deadlock loop or it has stalled at an input buffer. When an output buffer becomes available, packets in the multiqueue have priority over packets in the input buffers and, if more than one packet in the multiqueue desires the channel, it is allocated using a fair policy. If a packet must be moved into the multiqueue when it is full, a packet in the queue is chosen at random and *derouted* to the next available output buffer along a potentially non minimal path. As suggested in [Bol93], we utilize a multiqueue with five packet buffers.

4.4 SMART

This model is evaluated in the SMART (Simulator of Massive ARchitectures and Topologies) environment [Pet95]. Implemented in C++, SMART is an object-oriented discrete-event simulation tool for evaluating massively parallel architectures. Configuring some shell scripts, it is possible to select the node internal structure, the network topology, the internal router policies. The simulator allows the definition of the network topology, routing algorithm, packet length, number of virtual channels and buffers for both input and output lanes. Also, it is possible to monitor several metrics and time-dependent events, that are gathered in trace files. At the moment, SMART supports three families of topologies: k-ary n-cubes, k-ary n-flies and k-ary n-trees.

The parameters used by SMART are shown in Table 1. We model only the data structures of the FFT algorithms in the memory hierarchy. The experimental results in the following sections represent a somewhat larger cache. Also, the cache set size is varied between 1 and 16 in the analysis of the sequential algorithms, but it is set to 4 in all the parallel versions.

Parameter	**Meaning**	**Value**
$T_{compute}$	Computation of butterfly node	4
T_{read}	Read hit	3
T_{write}	Write hit	1
T_{read_miss}	Read miss	26
$T_{interrupt}$	Interrupt overhead	8
T_{packet}	Message marshaling overhead	20
$T_{routing}$	Routing delay experienced by a header flit	1
T_{link}	Link delay experienced by a generic flit	1
N_{flit_size}	Flit size in bytes	1
N_{sets}	Number of sets of the cache memory	1024
N_{set_size}	Set size or degree of associativity	4
N_{line_size}	Cache line size in bytes	64
N_{cache_size}	Cache capacity in bytes	262144

Table 1: SMART Parameters.

5 Experimental results

The speedup of a parallel algorithm should be estimated w.r.t. the best sequential algorithm. Thus, we first determine the best of the three algorithms proposed in Section 2.

As expected, the naive algorithm has a poor performance. Figure 4 a) shows that with a low degree of associativity, even small butterflies with more than 16384 input nodes suffer from cache conflicts. During the computation of the initial columns, both operands and the result of a butterfly node are stored on the same cache line with a direct mapped cache and we can have up to three (two read + one write) cache misses.

The hierarchical version share the same problem, though the performance degradation appears later, with butterflies whose input size is bigger than 65536, as shown in Figure 4 b).

The cache oriented version is not sensitive both to cache associativity and butterfly size, and will be used in the speedup evaluation. Figure 4 d) compares the three algorithms on a 4-ways associative cache.

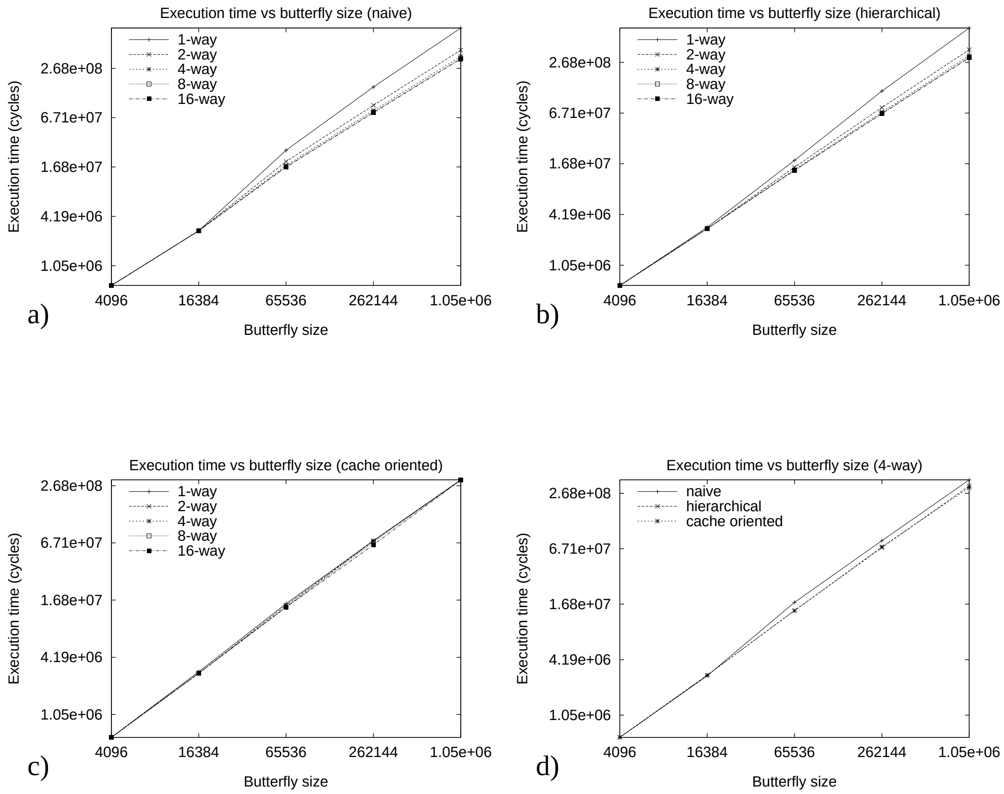

Figure 4: Sequential algorithms and cache associativity: a) naive b) hierarchical c) cache oriented. d) A comparison of the three sequential algorithms with a 4-way associative cache. All graphs share a double logarithmic scale.

In our experiments we mapped a 65536-input butterfly [4] on machines with up to 256 nodes. On the 4-ary n-trees both the basic and the overlapped algorithms have good speedups, as shown in Figure 5 a).

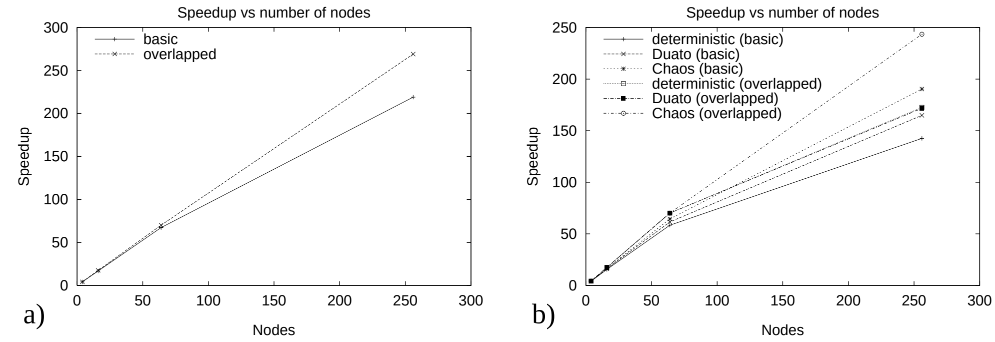

Figure 5: Speedups of the basic and overlapped parallel algorithms: a) 4-ary n-trees and b) 2-dimensional toroidal cubes with up to 256 nodes.

In particular, the overlapped one achieves superlinear speedups [5]. This is due to the favorable computation/communication ratio of the transpose algorithm, to smaller working sets that leads to a better memory hierarchy utilization, and to the communication performance of the interconnection network, whose bisection width scales linearly with the number of nodes. In fact the AAPB can be implemented as a congestion-free pattern in the k-ary n-trees [Hel94].

k-ary n-cubes are bisection-bandwidth-limited and are a significant test bed for our experiments. Figure 5 b) reveals that the speedups are very sensitive to the degree of overlapping and to the routing algorithm. Both versions of the Chaos router provides the best speedups, Duato's show an intermediate behavior and the basic deterministic version lags behind. The only surprise is Duato's overlapped version, which has the same performance of the deterministic counterpart.

We will inspect these results with greater detail in the following sections looking at the behavior of several parameters during program execution. These include the number of active processors, the network utilization, i. e. the ratio of active communication links over the global number of links, the accepted bandwidth, i. e. the amount of information per unit of time transferred to the processing nodes since the beginning of the communication phase and some components of the packet latency. We distinguish the following latencies.

- The trailing flit of a packet opens a virtual path between the source and the destination. The *head* latency is the delay experienced by it to get to the network interface of the destination.

[4] A 65536-input butterfly is the minimum data set size supported by the transpose algorithm on a 256 nodes architecture.

[5] It is worth noting that the parallel version partitions the data set over the processing nodes. Even if a given butterfly cannot fit in a single cache memory, the sub-butterflies allocated on each node can be fully contained. This situation can lead to superlinear speedups.

- The *network* latency is the time spent by a packet in the network, from the insertion of the trailing flit at the source node till the reception of the tail flit at the destination.

- The *end to end* latency is the time needed to the deliver a packet from a source node to its destination, including queuing delays in the local FIFOs.

5.1 4-ary 4-tree

The communication pattern of the basic algorithm on the k-ary n-trees can be efficiently implemented as a sequence of congestion-free steps. Given the number of nodes P, we divide the AAPB in P steps[6]. In step i, node P_j sends one or more packets, according to the dimension of the allocated butterfly and the maximum packet size, to node $P_{(j \ xor \ i)}$. The global communication pattern sharply separates the two local phases as shown by the number of active processors and the network utilization in Figure 6 a) and b). The communication performance is limited by the injection rate: with 256 nodes each message contains the value of a single butterfly node and can be delivered by the network interface in about 10 cycles, while the overhead in the communication processor T_{packet} is about 20 cycles. When the same data set is mapped on 64 nodes packets are about four times bigger and the network utilization is close to the unity. The absence of conflicts can be observed from the head and network latency in Figure 6 d).

In the overlapped algorithm the values of the output nodes become available during the computation, hindering the utilization of the congestion free communication pattern shown above. Therefore, the AAPB is implemented with a cyclic shift in this case. It consists of $P-1$ steps: in the i-th step processor P_j sends to processor $(P_j + i) \ mod \ P$. We can see in Figure 6 e)-h) that most of the communication takes place during the first phase, even if the head latency is sometimes higher than in the previous case because there are some conflicts in the routing switches.

5.2 16-ary 2-cube

We now turn our attention to the performance of the overlapped algorithm on a bi-dimensional torus with 256 nodes. The low network utilization of the deterministic routing algorithm, around 20%, limits the degree of overlapping (Figure 7 a)-d)). As can be be seen by the number of active processors, there is a gap 30000 cycles wide between the end of the first phase and the beginning of the second phase. The packets spend most of their latency waiting on the network interface at the source node. The end to end latency rises to 9000 cycles, two orders of magnitude more than the base latency. On the other hand, the accepted bandwidth remains constant, with an initial spike caused by the cyclic shift that sends the first message to neighboring nodes. While the latency of a single packet is difficult to estimate, the global behavior of the communication pattern is stable and predictable.

[6]But only $P-1$ steps require remote communication in each node.

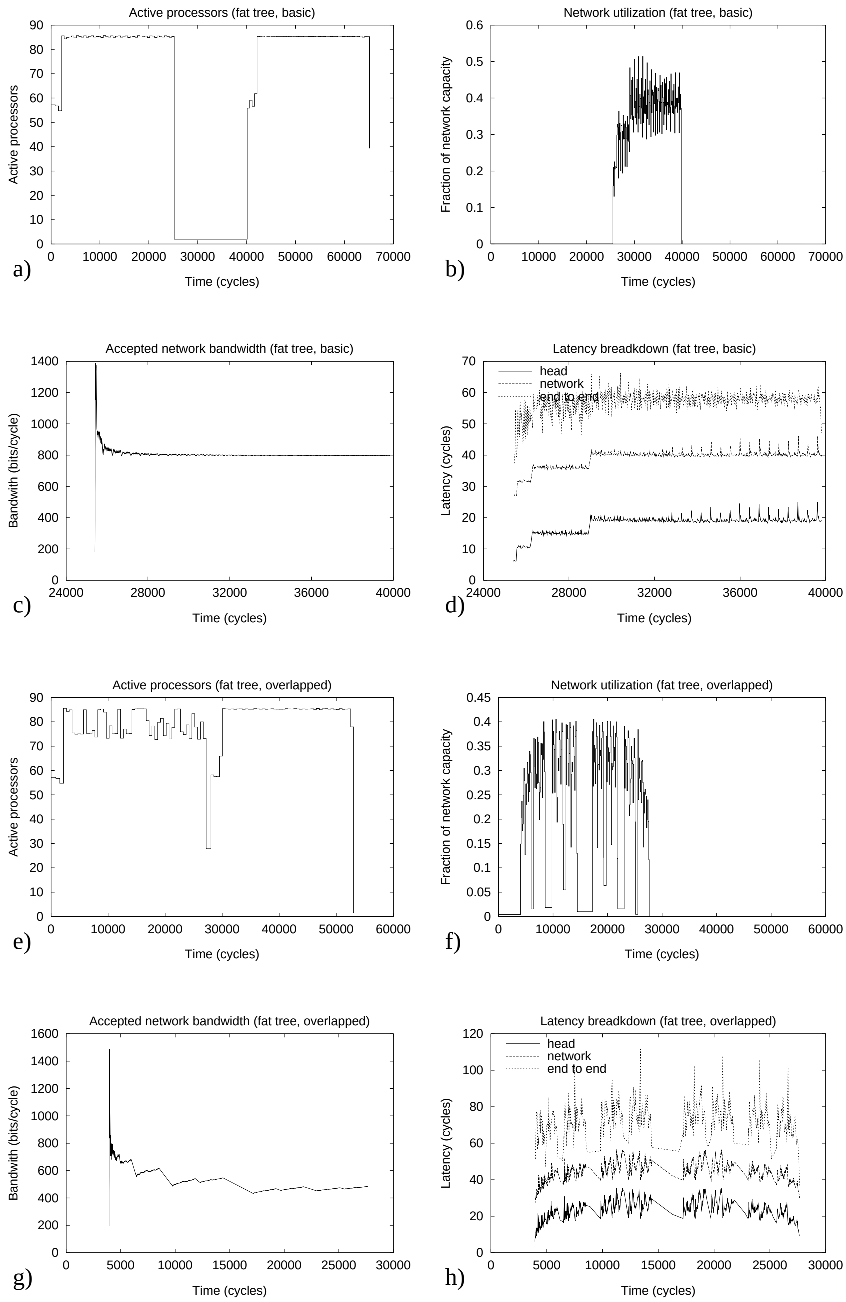

Figure 6: 4-ary 4-tree: a)-d) Basic algorithm. e)-h) Overlapped algorithm.

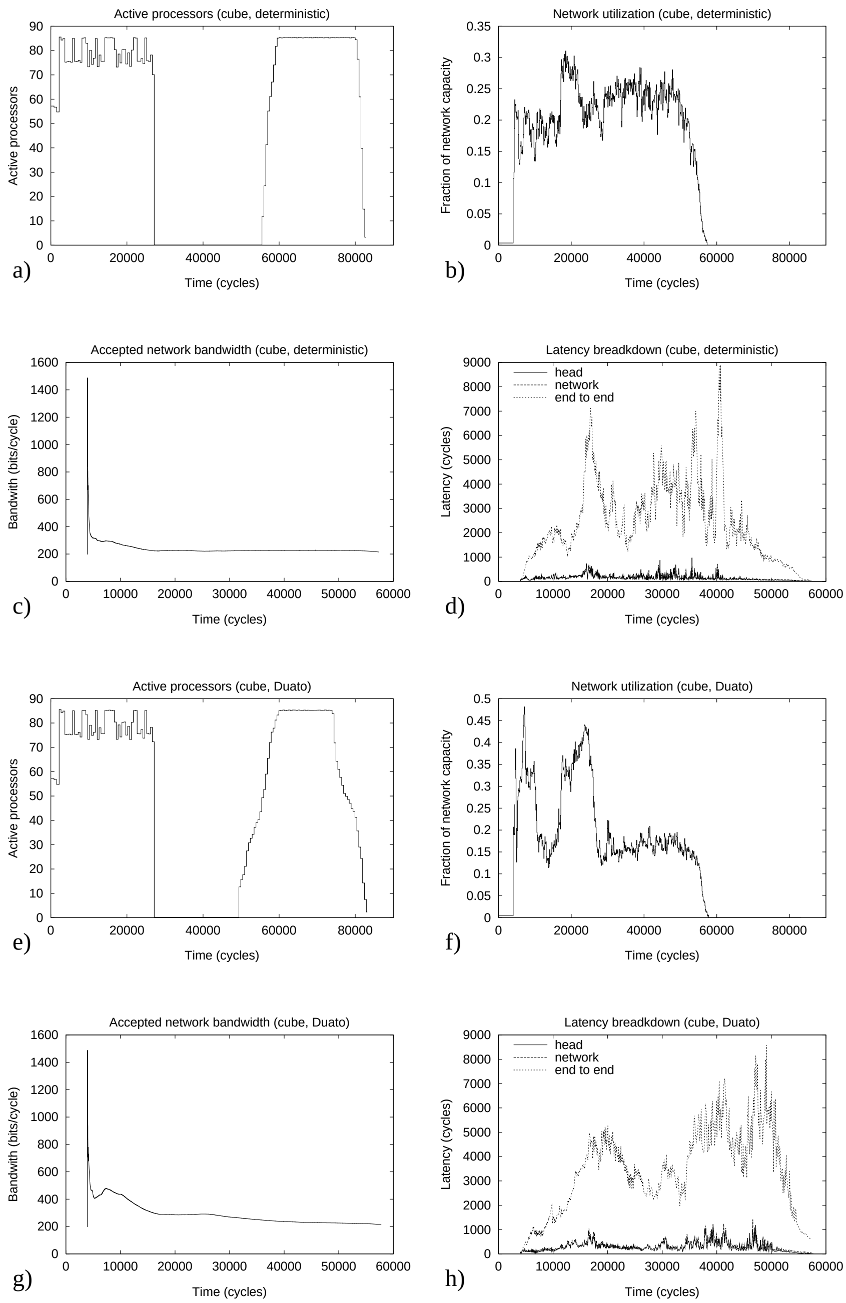

Figure 7: Toroidal 16-ary 2-cube, overlapped algorithm: a)-d) Deterministic routing. e)-h) Duato's routing.

Figure 7 f) explains the unexpected performance degradation of the algorithm based on Duato's methodology. The network utilization jumps at 45% two times and then drops at 15%, less than the deterministic algorithm. This algorithm has post-saturation problems and after the saturation point the accepted bandwidth drops after some oscillations. Newer algorithms based on the same methodology, as Disha [VPD95], use all the virtual channels for adaptive routing and replace the "escape channels" with a single central buffer dedicated to deadlock recovery rather than deadlock prevention.

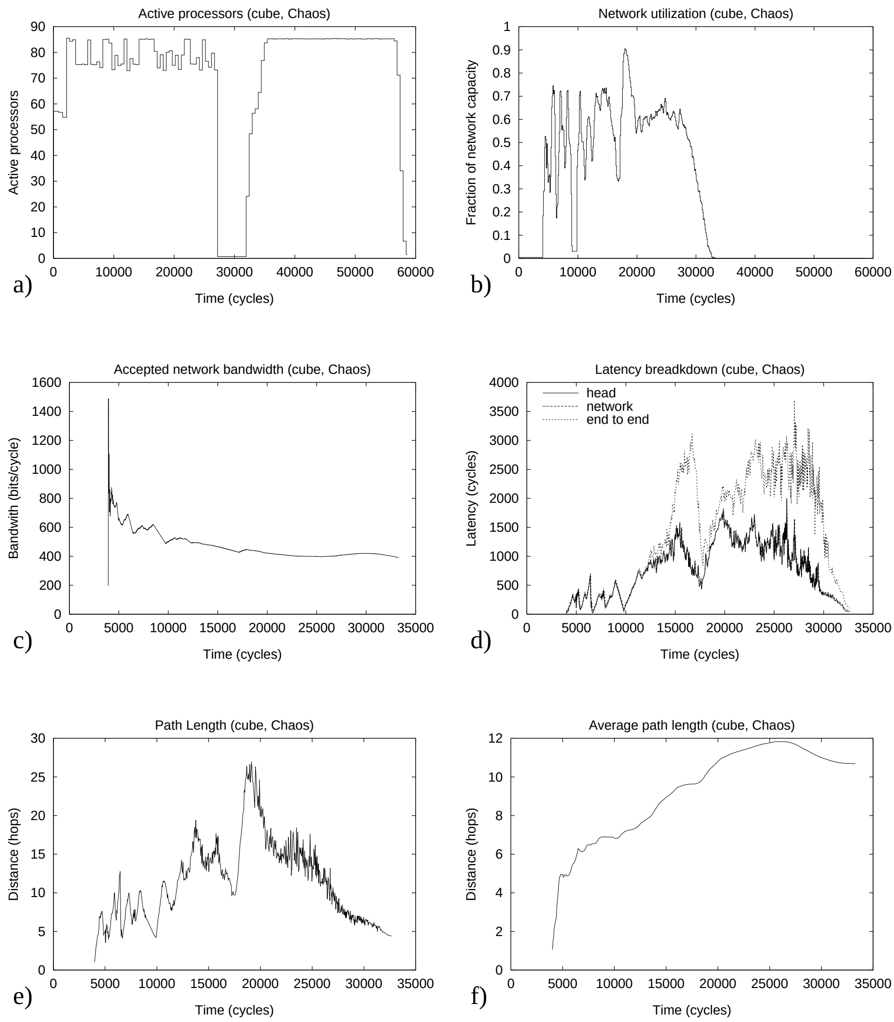

Figure 8: Toroidal 16-ary 2-cube, overlapped algorithm, Chaos routing.

The Chaos router reduces the gap between the two phases at few thousands cycles, as can be seen in Figure 8. The network utilization is on the average 60%, far more than the previous two algorithms. Anyway, the performance figures are not stable during the execution of the AAPB. The network utilization oscillates a few times between 70% and 20% and in one short span of time it falls down to 3%. A network that adopts cut-through as flow control strategy acts as a big buffer: also, the internal multiqueue provides five more packet buffers for each node. In the presence of saturated traffic, the network accepts as many packets as it can and then suddenly blocks and becomes instable. In our example this happens in a short critical interval, after 9000 cycles from the start: the network utilization falls down and no packets are delivered for a while. The recovery from this situation is obtained by derouting many packets, that are sent away from their minimal path. Figure 8 e) shows that the path length after the critical interval reaches 25 hops, three times more than the topological average distance.

6 Discussion

The task of evaluating the network performance is not an easy one, but it is crucial to develop efficient parallel algorithms. The experimental results shown above suggest that the communication patterns can be roughly divided into two classes: *latency-bound* and *bandwidth-bound*. A latency-bound pattern is limited by the critical path of the communication while a bandwidth-bound pattern is limited by the network capacity or the injection bandwidth. Alternatively, a pattern is bandwidth-bound if it would benefit from providing more bandwidth.

A latency-bound pattern is generated by the overlapped version on the fat-tree: in this case the gap between the two phases is determined by the base network latency of the final non-overlapped communications in the cyclic shift.

All the remaining patterns are bandwidth-bound, included the basic version of the fat-tree, which is limited by the overhead of the network interface rather than routing and network characteristics. Many theoretical models, but also some research results [CLR94], are based on the assumption that latency can be modeled with a constant value. In the presence of bandwidth-bound patterns this parameter is not easy to to predict. Also, it can be some order of magnitudes bigger than the base latency without conflicts [BW95]. If we switch from the latency of the single packet to the global behavior of the network, we can note that the accepted bandwidth remains stable during the AAPB. After an initial period of 10000 cycles of deterministic routing, the processing nodes receive a constant amount of data per unit of time. Starting from the total amount of information exchanged during the communication pattern we can give a reasonable approximation of its execution time. The other two adaptive routing algorithms are less stable, but nevertheless the accepted bandwidth lies within a narrow range in both cases. We think that, in order to understand and predict the network performance of bandwidth-bound patterns, the routing algorithms should guarantee a reliable behavior after saturation.

The main goal of this paper is to analyze the network characteristics rather than to compare routing algorithms. A fair comparison between Duato's methodology and the Chaos router should take into account the router complexity. The internal architecture of the Chaos router is extremely more complicated and would probably imply an increased delay in clock cycle and in packet handling.

7 Conclusion

In this paper we have analyzed in depth the behavior of two important classes of interconnection networks with up to 256 nodes, the k-ary n-trees and the k-ary n-cubes, during the execution of some parallel FFT algorithms.

The basic FFT transpose algorithm can be efficiently mapped on the k-ary n-trees, because the communication pattern generated by the personalized broadcast can be factorized in a sequence of congestion-free steps. The overlapped algorithm, that interleaves communication with computation, exploits the presence of a communication processor and reaches a superlinear speedup. This is due to the smaller working sets of the parallel algorithm w.r.t. the sequential one and to the high degree of overlapping that enables the delivery of most packets during the first computational phase.

Though k-ary n-cubes are less favored than the trees in terms of bisection bandwidth, the overlapped algorithm in conjunction with an efficient routing algorithm narrows the gap between the two families of interconnection networks. The non-minimal adaptive routing, the Chaos router, has the best speedup, while the other two algorithms, the minimal adaptive based on Duato's methodology and the deterministic, exploits less network bandwidth. Both adaptive algorithms suffer from post-saturation problems. After an initial period, the network utilization begins to oscillate and drops down to 15% during the execution of the non-minimal algorithm. The Chaos router alternates over a short span of time values of the network utilization between 70% and 3%.

In the final part of the paper we have outlined a rough classification that divides the communication patterns in latency and bandwidth-bound. While the latency of the single packet in a bandwidth limited pattern is difficult to determine, the global communication bandwidth falls in a restricted range and tends to an asymptotic fixed value when the routing algorithm is stable in the presence saturated traffic. This happens with the deterministic routing of the cubes and with the adaptive routing of the fat-trees. Steady state bandwidth can be used to estimate the completion time of a given global communication pattern and should be an important goal for designers of routing algorithms.

References

[ACS89] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. On Communication Latency in PRAM Computation. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 11–21, 1989.

[ACS90] Alok Aggarwal, Ashok K. Chandra, and Marc Snir. Communication Complexity of PRAMs. *Theoretical Computer Science*, pages 3–28, March 1990.

[AHS93] James Aspnes, Maurice Herlihy, and Nir Shavit. Counting Networks. Technical Report CRL 93/11, Digital Equipment Corporation, Cambridge Research Lab, August 1993.

[AISS95] Albert Alexandrov, Mihai F. Ionescu, Klaus E. Schauser, and Chris Scheiman. LogGP: Incorporating Long Messages into the LogP Model - One step closer towards a realistic model for parallel computation. Technical Report TRCS 95-09, Department of Computer Science, University of California, Santa Barbara, CA, April 1995.

[Bai87] David H. Bailey. A High-Performance Fast Fourier Transform Algorithm for the Cray-2. *Journal of Supercomputing*, 1:43–60, 1987.

[Bai88] David H. Bailey. A High-Performance FFT Algorithm for Vector Supercomputers. *International Journal of Supercomputer Applications*, 2:82–87, 1988.

[Bai90] David H. Bailey. FFTs in External or Hierarchical Memory. *Journal of Supercomputing*, 4:23–35, 1990.

[BNK92] Amotz Bar-Noy and Shlomo Kipnis. Designing Broadcasting Algorithms in the Postal Model for Message-Passing Systems. In *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures*, pages 13–22, June 1992.

[Bol93] Kevin Bolding. *Chaotic Routing: Design and Implementation of an Adaptive Multicomputer Network Router*. PhD thesis, University of Washington, Department of Computer Science and Engineering, Seattle, WA, July 1993.

[BW95] Douglas C. Burger and David A. Wood. Accuracy vs. Performance in Parallel Simulation of Interconnection Networks. In *International Symposium on Parallel Processing*, April 1995.

[Chu87] Clare Chu. Load Balanced FFT Implementation on the Intel iPSC. Technical Report 87-824, Department of Computer Science, Cornell University, Ithaca, New York, March 1987.

[CKP+93a] David E. Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, San Diego, CA, May 1993.

[CKP+93b] David E. Culler, Richard Karp, David Patterson, Abhijit Sahay, Klaus Erik Schauser, Eunice Santos, Ramesh Subramonian, and Thorsten von Eicken. LogP: Towards a Realistic Model of Parallel Computation. In *Proceedings of the Fourth ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, pages 1–12, San Diego, CA, May 1993.

[CLMY96] David Culler, Lok Tin Liu, Richard P. Martin, and Chad Yoshikawa. LogP Performance Assesment of Fast Network Interfaces. *IEEE Micro*, February 96. to appear.

[CLR94] Satish Chandra, James R. Larus, and Anne Rogers. Where is Time Spent in Message-Passing and Shared-Memory Programs ? In *Proceedings of the Sixth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS VI)*, October 1994.

[D+92] William J. Dally et al. The Message-Driven Processor. *IEEE Micro*, pages 23–39, April 1992.

[Dal90] William J. Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[Dal92] William J. Dally. Virtual Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, March 1992.

[DS87] William J. Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, C-36(5):547–553, May 1987.

[Dua94] Josè Duato. A Necessary and Sufficient Condition for Deadlock-Free Adaptive Routing in Wormhole Networks. In *International Conference on Parallel Processing*, volume I - Architecture, pages 142–149, 1994.

[FW78] S. Fortune and J. Willie. Parallelism in Random Access Machines. In *Proceedings of the 10th Annual Symposium on Theory of Computing*, pages 114–118, 1978.

[Get92] Vladimir Getov. 1-Dimensional Parallel FFT Benchmark on SUPRENUM. In D. Etiemble and J. C. Syre (Eds.), editors, *PARLE'92, Parallel Architectures and Languages Europe*, pages 163–174, 1992.

[GK93] Anshul Gupta and Vipin Kumar. The Scalability of FFT on Parallel Computers. *IEEE Transactions on Parallel and Distributed Systems*, 4(8):922–932, August 1993.

[Hel94] Steve Heller. Congestion-Free Routing on the CM-5 Data Router. In Kevin Bolding and Lawrence Snyder, editors, *First International Workshop, PCRCW'94*, volume 853 of *LNCS*, pages 176–184, Seattle, Washington, USA, May 1994.

[Hwa93] Kai Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*. McGraw-Hill, Inc., 1993.

[KSS92] Richard Karp, Abhijit Sahay, and Eunice Santos. Optimal Broadcast and Summation in the LogP Model. Technical Report UCB/CSD 92/721, Computer Science Division, University of California, Berkeley, 1992.

[Lei92] F. Thomson Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, San Mateo, CA, USA, 1992.

[MG66] Gentleman W. M. and Sande G. Fast Fourier Transform - For Fun and Profit. *AFIPS Proceedings*, 29:563–578, 1966.

[Pet95] Fabrizio Petrini. SMART: a Simulator of Massive ARchitectures and Topologies. internal note, March 1995.

[PV95a] Fabrizio Petrini and Marco Vanneschi. k-ary n-trees: High Performance Networks for Massively Parallel Architectures. Technical Report TR-18/95, Dipartimento di Informatica, Università degli studi di Pisa, December 1995.

[PV95b] Fabrizio Petrini and Marco Vanneschi. Physical Architectures: Interconnection and Communication. In Gianfranco Balbo and Marco Vanneschi, editors, *General Purpose Parallel Computers - Architectures, Programming Environments, and Tools*, chapter 2, pages 27–65. Edizioni ETS, Pisa, September 1995.

[PY88] C. H. Papadimitriu and M. Yannakakis. Towards an Architecture Independent Analysis pf Parallel Algorithms. In *Proceedings of the 20th Annual Symposium on Theory of Computing*, pages 510–513, 1988.

[RWW91] Herman Roebbers, Peter Welch, and Klaas Wijbrans. A generalized FFT algorithm on transputers. Technical report, University of Kent at Canterbury, UK, 1991.

[Sah92] Abhijit Sahay. Hiding Communication Costs in Bandwidth-Limited Parallel FFT Computation. Technical Report UCB/CSD 92/722, Computer Science Division, University of California, Berkeley, 1992.

[Swa84] P. N. Swarztrauber. FFT Algorithms for Vector Computers. *Parallel Computing*, 1:45–63, 1984.

[Swa87] P. N. Swarztrauber. Multiprocessor FFTs. *Parallel Computing*, 5:197–210, 1987.

[Val90] Leslie G. Valiant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8):103–111, August 1990.

[VPD95] Anjan K. V., Timothy Mark Pinkston, and Josè Duato. Deadlock-Free Adaptive Wormhole Routing with Disha Concurrent. Technical Report CENG-95-21, Electrical Engineering - Systems Department, University of Southern California, November 1995.